Introduction to Languages

Mattox Beckman
beckman@iit.edu

Illinois Institute of Technology

Welcome to CS 440!!

Your goal for this lecture is to. . .

- know who your instructors are, and how they plan to organize the course.
- know what you should expect to learn in this course.
- know why we study programming languages.
- know two questions to use to categorize programming languages, and the answers to those questions for your favorite languages.
- know the major themes of this course.
- know a few things about the language we will use.

Instructor Mattox Beckman <beckman@iit.edu>

Office 110 SB

Office Hours Monday and Wednesday, 14.00–15.00; and by appointment.

Home Page http://www.cs.iit.edu/~beckman

TAs

- TBA.

Course Web Page http://prancingTarantula.net/cs440-sp07 You are expected to check the announce page daily.

Question Database http://prancingTarantula.net/qdb

Language OCaml, a dialect of ML. See http://caml.inria.fr/ocaml

Text The required text is Jason Hickey's Introduction to the Objective Caml Programming Language. It will often be called the Caltech Book. A recommended resources is the *Dragon Book*—Aho, Sethi, Ullman "Compilers: Principles, Techniques, and Tools".

Contacting Your Instructors

Google Group: We will use Google Group `iit-cs440-2007-01` for the discussion features. Post questions of general interest to the class.
"When is the next machine problem due?", "Does anyone know where I can find OCaml for my PC?", "I keep getting this error, anybody have an idea what's wrong?"

Email: good for urgent things, individualized requests, etc.
"I'd like to make an appointment to discuss my code.", "I think there's an error in my grade."

Phone: for very urgent things. "I'm going to miss the exam."

Responsibilities

Our job is to provide an "optimal learning environment".

- Assignments will be clearly written and administered.
- Questions will be answered in a timely fashion.
- Objectives of lectures and assignments will be clearly communicated.
- Grades will be fair, meaningful, and reflect mastery of course material.
  - C grade means "can reliably recognize the correct answer"
  - A grade means "can reliably generate the correct answer"

Who Does What

- Instructor
  - Writes Lectures, Exams, most MPs
  - Sets Grade cutoffs and course policy
- TAs
  - Machine Problem Administration
  - Grading
  - Discussion board, "first line of contact"

Your Responsibilities

- Check the course web page and course news group daily.
- Do the programming assignments in order to learn them.
- Attend lectures — or find out what happened there.
- **Take responsibility and initiative in learning material** — experiment!

You are the one primarily responsible for your education.

Lectures

- Best research indicates that students learn better when lectures include in-class activities.
  - Bring pen and paper.
  - You can talk about the example problems during the activity time, but this is not an invitation to talk about other things, or to talk during the rest of the lecture.
  - Answers will be included in the on-line slides, but not handed out.
  - **Not optional** — you will be tested on this material.

Machine Problems

- 7 Machine Problems — collectively worth 20% “Participation grade”
- Designed to help you study for the exams.
- Full collaboration allowed.
- Don’t use the “perturbation method” of solving machine problems! We expect you to *understand* the solution and the process very well.
- Multiple MPs will be active at a time. See the schedule.

Exams

- The purpose of an exam is to measure mastery of material.
- 2 midterm exams — worth 20% each.

  First Date February 19, 2007

  Second Date April 9, 2007
- MP material will be tested!
- Final exam — worth 30%
  - Wednesday, May 9, 2007 14:00–16:00
  - It will be non-cumulative. Sort-of.

Grade Guarantees

The course will not be graded on a curve or by ranking. Instead, we have the following grade guarantees:

- 85% A
- 70% B
- 55% C
- 40% D
- In the event that something goes wrong, we may lower the curve.

Themes

The course has three major themes:

1. Languages
   What kinds of things can we say in a language?
2. Parsing
   How do we get the computer to read what we said?
3. Interpreting and Compiling
   How do we get the computer to do what we said?

Two Fundamental Questions

- Question: Why should we study programming languages?
- There are six[a] major classes of languages:

1. Low Level
2. Imperative
3. Data encapsulating
4. Object Oriented
5. Functional
6. Logic

What is a program?
What is the nature of data?

[a] Actually, there are a whole lot more than that. This is just how I'm dividing it up.

Low level Languages

- A program is a *series of control signals for a CPU*
- Data consists of integers, memory address, and IEEE floating point
- Based on the architecture of the CPU
- Usually not very expressive
- Examples: Assembly languages, microcode

```
1 L1: MUL AX,2  ; double A
2     DEC BX    ; decrement B
3     JZ  L1    ; repeat until B is zero
```


Imperative Languages

- A program is a *list of commands to be executed*
- Data consists of low-level types (like integers) and composite types (like records and arrays)
- Examples: C, BASIC, Fortran, Forth, Pascal

```
1 while (b > 0) {
2    a = a * 2;
3    b = b - 1;
4 }
```

Data-encapsulation languages

- A program is a *list of commands to be executed*
- Data consists of low-level types and composite types, and we can hide the details from the user.
- Examples: Clu, Modula 2, Ada

```
Poly = cluster is create, degree, coeff, ....
   rep = record[coeffs: array[int], li, hi: int]
   create = proc(c,n : int) returns (Poly)
      A : array[int] := array[int]$create(0)
      A[n] := c
      return(up(rep${coeffs: A, lo: n, hi: n}))
   end create
```


Object Oriented Languages

- A program is a *list of commands to be executed*
- Data consists of *objects* which can send and receive messages. — an object is “functions with state”
- Examples: Smalltalk, C++, Java

```
class Square {
public:
  int x,y,h,l;
  Square() { x = y = h = l = 0; }
};
```


Functional Languages

- A program is an *expression to be evaluated*
- Data consists of low level types and “higher order types”—functions.
- Examples: Lisp, Scheme, ML

```
# let twice f x = f(f(x));
# let inc x = x + 1;
# inc 5;
6
# twice inc 5;
7
```


Logic Programming Languages

- A program is a *logical predicate to satisfy*
- Data consists of a set of assertions about what we know to be true.
- Example: Prolog

```
-  human(socrates).
-  mortal(X) :- human(X).
-? mortal(Who).
Who = socrates
```

So, what should you learn?

- Understand major classes of programming languages: techniques, features, styles.
- How to specify formally the meaning of a language — to people and to the computer.
- Three Powerful Ideas:
  1. Recursion
  2. Abstraction
  3. Transformation
- How to choose a language.
- How to implement a language.

Emphasis: learn theory and apply it.

How am I going to learn it?

There are two approaches to teaching a PL course.

- Approach 1: “Language of the Month Club”
  - Lots of time spent on syntax, fundamentals tend to get lost.
  - You’ll forget them all anyway.
- Approach 2: “Host Language”
  - Learn one language, use it to write interpreters for all the other languages.
  - You actually get to see how a language is put together.

So, which language to use?

Introducing OCaml

- Assumption 1: You all know a lot about Imperative/OO languages.
- Assumption 2: Few or none of you know anything about functional languages.
- Solution: Emphasize functional languages
- Benefit: Functional languages are good at modeling other languages.

The host language for 440 is Objective Caml.

Features of OCaml

Language Features:

- higher order functional language
- call by value parameter passing style
- modern syntax
- parametric polymorphism
- automatic garbage collection

And two more things....

- It’s very fast — the winners of the 2004, 2000 and 1999 ICFP Programming Contests used OCaml.
- The error messages make sense.

Colophon

- Slides are typeset using LaTeXand the *Prosper* document class.
- Output is in PDF.
- Example CLU code from Sam Kamin's *Programming Languages: An Interpreter Based Approach*